UNITED STATES PATENT APPLICATION

FOR

METHOD AND APPARATUS FOR SCHEDULING FOR PACKET-SWITCHED NETWORKS

BY

SHIH-CHIANG TSAO,

YING-DAR LIN,

HAI-YANG HUANG,

AND

CHUN-YI TSAI

Reference to Related Applications

[001] This application claims priority from co-pending prior provisional application Serial No. 60/253,930, filed November 30, 2000 for "PRE-ORDER DEFICIT ROUND ROBIN: A NEW SCHEDULING ALGORITHM FOR PACKET-SWITCHED NETWORKS."

Field of the Invention

[002] This invention relates generally to packet scheduling. In particular, the invention relates to a packet scheduling method and apparatus for packet switched networks.

Background of the Invention

[003] In recent years, many packet scheduling algorithms have been proposed to reduce congestion, minimize delay (i.e., latency), and maintain fairness, especially to accommodate a high number of packet flows. Unfortunately, many of these algorithms can only be applied to fixed-size packets. Furthermore, many of these algorithms (even if they do not require fixed sized packets) exhibit poor performance as the number of packet flows increases.

[004] Deficit Round Robin (DRR) is an algorithm which allows for variable-sized packets. Under DRR, a node rotationally selects packets to send out from all flows that have queued packets. During a round, each flow accumulates credits in discrete increments (e.g., in bytes) called a quantum. Unfortunately, DRR typically requires a quantum value that is very large, i.e., many times the size of the

maximum packet size for a flow. The data presented in Table 1 below illustrates the above problem.

TABLE 1
THE TRAFFIC PARAMETERS AND *QUANTUM* SIZE OF 4 FLOWS

Flow ID	Reserved Rate (Mbps)	Traffic Type	Maximum Packet Size (byte)	Ratio of Max Packet Size to Reserved Rate	Quantum Size (byte)
A	12 8	CBR	400	250	512
B	16	CBR	640	320	640
C	64	CBR	800	100	2560
D	64	CBR	100	12 5	2560

[005] The data in Table 1 assumes four flows, sharing the same link, and a link capacity of 160 megabits per second. As illustrated in Table 1, the quantum size of a flow can reach very large values relative to the flow's maximum packet size. For example, flow *D* has a quantum size of 25.6 times (2560/100) the maximum packet size. Unfortunately, due to the large quantum size required by DRR, DRR's performance can be poor compared to other algorithms such as Self-Clocked Fair Queuing (SCFQ) and Weighted Fair Queuing (WFQ).

[006] However, simply reducing the quantum size also creates problems and is generally ineffective. For example, reducing the quantum size can cause a node using DRR to select no packets to send out after querying all flows in a particular round. This causes unacceptable delay and, again, causes poor performance. Thus, simply reducing the quantum sized used in a DRR node is generally not effective. Accordingly, it would be desirable to provide a scheduling algorithm and apparatus which does not require fixed size packets and exhibits good performance, including when there is a high number of packet flows.

SUMMARY OF THE INVENTION

[007] In accordance with the invention, a method for scheduling a packet, comprises: receiving a packet; identifying a flow for the packet; classifying the

packet based on the identified flow; and buffering the packet in one of a plurality of queues based on the classification of the packet.

[008] In accordance with another aspect of the present invention, a system for scheduling a packet, comprises: an input to receive a plurality of packet; an arrival module to identify a flow for each of the plurality of packets; a classifier to assign each of the plurality of packets to one of a plurality of queues based on the identified flow; a server for selecting one of the plurality of queues based on a hierarchical order; and an output for outputting a packet from the selected queue.

[009] In accordance with yet another aspect of the present invention, an apparatus for scheduling a packet, comprises: means for receiving a packet; means for identifying a flow for the packet; means for classifying the packet based on the identified flow; and means for buffering the packet in one of a plurality of queues based on the classification of the packet.

[010] Additional advantages of the invention will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practice of the invention. The advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims.

[011] It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention, as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

[012] The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate embodiments of the invention and together with the description, serve to explain the principles of the invention. In the drawings:

[013] Fig. 1 illustrates a node 100 utilizing a Pre-Order Deficit Round Robin (PDRR) architecture in accordance with principles of the present invention;

[014] Fig. 2 shows a method for scheduling packets in accordance with principles of the present invention;

[015] Fig. 3 shows a method of transmitting packets in accordance with principles of the present invention;

[016] Fig. 4a illustrates the operation of Deficit Round Robin (DRR) in comparison with weighted fair queuing (WFQ);

[017] Fig. 4b shows the present invention using PDRR operating on the same input pattern and assumptions used in Fig. 4a;

[018] Figs. 5-8 show various simulation results to compare the performance of embodiments consistent with the present invention using PDRR with DRR and SCFQ; and

[019] Fig. 9 shows performance consistent with the present invention as the number of priority queues is varied for a specific traffic environment.

DESCRIPTION OF THE EMBODIMENTS

[020] Reference will now be made in detail to exemplary embodiments of the invention, examples of which are illustrated in the accompanying drawings.

Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

[021] Embodiments consistent with the present invention provide pre-order deficit round robin (PDRR) architecture to execute a scheduling algorithm which minimizes delay while maintaining fairness in a packet switched network. Embodiments consistent with the present invention use O(1) per-packet time complexity in most cases (i.e., as the number of packet flows increases) and is amenable to variable-length packets.

[022] Analysis results from testing embodiments consistent with the present invention with respect to three measures, including latency, fairness and per-packet time complexity are also provided. The analysis results provide supporting evidence that embodiments consistent with the present invention offer better performance in latency, fairness and lower time complexity. Furthermore, simulation results are provided to demonstrate the behavior of embodiments consistent with the present invention.

[023] Fig. 1 illustrates a node 100 utilizing the PDRR architecture in accordance with principles of the present invention. In particular, node 100 comprises an input port 102, a processor 104, a packet arrival module 106, a pre-order queuing module 108, a packet departure module 110, and an output port 122.

[024] Input port 102 interfaces node 100 to a link, e.g., to other nodes (not shown) and receives incoming packets. For purposes of illustration, node 100 is shown with one input port, i.e., input port 102. However, node 100 may be implemented with any number of input ports for receiving incoming packets.

[025] Processor 104 performs various operations for receiving, scheduling, and passing packets. Processor 104 may be implemented using hardware logic in combination with software and an operating system. Examples of the operating system and software include the UNIX operating system and the LINUX operating system for executing code written using C and C++.

[026] Packet arrival module 106 receives packets from input port 102, identifies each packet's flow, and places each packet in its corresponding flow queue. Packet arrival module 106 determines the number n and identification of flow queues 112_1-112_n based upon information received from packet departure module 110 via path 124. Packet arrival module 106 may also provide notification, e.g., to packet departure module 110 via processor 104, when a packet arrives for a new flow to be serviced by node 100. Furthermore, packet arrival module 106 may notify pre-order queuing module 108, e.g., if there are no other packets for a particular flow. As shown in Fig. 1, packet arrival module 106 comprises a set of flow queues 112-112_n, where n is the number of flows currently being serviced by node 100. Packet arrival module 106 may be implemented using any combination of hardware logic and software. Packet arrival module 106 may use processing functions of processor 104 to execute instructions in software. Below is one example of pseudo-code called "*PKT_Arrival*" which may be used by packet arrival module 106 to place each packet into its corresponding flow *Fq* (i.e., one of flow queues 112_1-112_n).

```
PKT_Arrival module
  i←ExtractFlow(p)              // Get the flow # of packet p
  Enqueue(p, Fqi)
  If NumItem(Fqi)=1 Then        // The equal implies that Fqi is empty before p
     SendMsg(PKT_Pass, i)       was placed into it, and need to notify
                                PACKET_Pass to handle the flow i
```

[027] Output port 122 outputs packets passed from node 100 on to a link, e.g., to other nodes. For purposes of illustration, node 100 is shown with one output port, i.e., output port 122. However, node 100 may be implemented with any number of output ports. In addition, node 100 may be implemented with dual purpose ports which function as both an input and output port.

[028] Pre-order queuing module 108 places packets from non-empty flow queues 112-112n into a second set of queues. Pre-order queuing module 108 may process any size packet. Pre-order queuing module 108 comprises a classifier sub-module 114 and a priority queuing sub-module 115.

[029] Classifier sub-module 114 retrieves packets from each non-empty flow queue, i.e., flow queues 112_1-112_n within packet arrival module 106, determines a priority for each packet, and places each packet in an appropriate priority queue, i.e., priority queues 116_1-116_z, within priority queuing sub-module 115. In addition, classifier sub-module 114 may enable a packet (e.g., for a high priority flow) to be considered for transmission immediately. Priority queuing sub-module 115 maintains a number of priority queues 116_1-116_z, where z is represents the number of priorities used by classifier sub-module 114.

[030] Pre-order queuing module 108 may implement classifier sub-module 114 and priority queuing sub-module 115 in a wide variety of ways. For example, pre-order queuing module 108 may use processing functions of processor 104 and execute instructions in software. Below is one example of pseudo-code called "*PKT_Pass*" which may be used by pre-order queuing module 108. *PKT_Pass* decides to which class *j* a packet belongs and places the packet into the

corresponding priority queue Pq_j (i.e., priority queues 116_1-116_z) from its Fq (i.e., flow queues 112_1-112_n).

```
PKT_Pass module
While(TRUE)
{I←WaitMsg()                          // Wait until a packet is placed to the empty Fq_I
 If Round_I≠Round_sys                 // Non-equivalent implies a new round is arrival
 {  Round_I←Round_sys
    DC_I←Max(DC_I, Quantum_I)
 }
 While DC_I>0 and
 NonEmpty(Fq_I)                       // Classify eligible packets into PQ
 {   PktSize←Size(Head(Fq_I))         // Get the size of the packet at head of Fq_I
     If (PktSize<DC_I) Then           // If flow I credits are enough to send out packets.
     {  DC_I←DC – PktSize             // Take out the used credits
        j←Z – (DC_I / Pqg_I)          // Compute the j
        Enqueue(Dequeue(Fq_I),        // Move the head packet of Fq_I to eligible Pq_j
        Pq_j)
        If NumItem(Pq_j)=1            //It implies that Pq_j is empty and j is non-existed in
        Then                          // the min heap before the last packt was placed
                                      into it
            MH_Insert( j)             //Insert j to min heap
     }
 }
 If NonEmpty(Fq_I) Then               // Imply the residual credits are not enough
 {  Enqueue(I, AckList)
    If NumItem(AckList)=1 Then SetEvent(EV_actlist)
 }
}
```

[031] Packet departure module 110 retrieves packets from non-empty priority queues, i.e., priority queues 116_1-116_z, within priority queuing sub-module 115 and outputs packets to output port 122. Packet departure module 110 comprises a server sub-module 118 and a service list 120.

[032] Server sub-module 118 services in a round-robin fashion each non-empty priority queue among priority queues 116_1-116_z. Server sub-module 118 refers to service list 120 to determine the non-empty priority queues. In one embodiment, server sub-module 118 declares a new "round" of service and services the non-empty priority queues among priority queues 116_1-116_z using, e.g., an algorithm similar to a deficit round robin (DRR) algorithm. Server sub-module 118 in conjunction with service list 120 may use a quantum and a deficit counter for each flow of packets to determine how a particular priority queue is serviced. The

quantum represents a share of available bandwidth (e.g., in bytes) allocated to a flow within the period of one round. A $quantum_i$ for a flow i can be expressed as

[033] $$Quantum_i = \frac{r_i}{C} \times F, \qquad (1)$$

[034] where r_i is the rate allocated to flow i, C is the link service rate, and F is the frame size that represents the summation of quantums for all flows.

[035] A flow accumulates shares of bandwidth (i.e., in quantum increments) during a round. The deficit counter accumulates any residual quantum of flow i in the (j-1)th round, which can be represented as $DeficitCounter_i^{j-1}$. The next time that flow i is serviced by a node, an additional $DeficitCounter_i^{j-1}$ bytes of data (i.e., incremented by $quantum_i$) can be sent out in the j th round. Server sub-module 118 verifies the size of the packet at the head of the priority queue ("head packet") currently being serviced. As described with reference to Fig. 3 below, server sub-module 118 also determines when a particular packet will be transmitted, e.g., via output port 122.

[036] Server sub-module 118 maintains service list 120. As noted above, service list 120 includes data for all flows currently being serviced by node 100. For each flow, service list 120 may include a flow identifier, a deficit counter, and a quantum. If a flow has no packets, e.g., within flow queues 112_1-112_n, server sub-module 118 may delete the flow identifier from service list 120. Alternatively, when a packet arrives for a new flow, server sub-module 118 may add an identifier for the new flow to service list 120.

[037] In one embodiment, server sub-module 118 updates the deficit counters ($DeficitCounter_i^j$) in service list 120 according to the equation:

[038] $DeficitCounter_i^j = DeficitCounter_i^{j-1} + Quantum_i$.

[039] As noted above, the quantum indicates the increments of shares of bandwidth accumulated by a particular flow. Server sub-module 118 calculates the quantum such that a packet can be processed in O(1) operations, The quantum for a flow may be larger than the maximum packet size within the flow so that at least one packet per backlogged flow can be served in a round. The quantum for any two flows *i* and *j* may be expressed by

[040] $$\frac{Quantum_i}{Quantum_j} = \frac{r_i}{r_j}. \qquad (2)$$

[041] Even assuming that all flows begin heavily backlogged at time *t*, the principles of the present invention allow server sub-module 118 to exhibit good performance. That is, server sub-module 118 can send out the packet with the earliest virtual finishing timestamp first among the packets at the heads of all flow queues. Under the above heavy backlog assumption, the virtual finishing timestamp of a packet may be computed as

[042] $$TS_i^m = TS_i^{m-1} + \frac{L_i^m}{r_i}, \qquad (3)$$

[043] where TS_i^m denotes the timestamp of the *m*th packet of flow *i* after time *t* and, for all *i*, TS_i^0 is set to zero at time *t*, r_i denotes the allocated rate of flow *i*, and L_i^m denotes the size of the *m*th packet of flow *i* after time *t*. Equation (3), by substituting Acc_i^m for $TS_i^m \times r_i$, is equivalent to

[044] $$\frac{Acc_i^m}{Quantum_i} = \frac{Acc_i^{m-1} + L_i^m}{Quantum_i}, \qquad (4)$$

[045] where Acc_i^m denotes the accumulated amount of data within a byte that flow *i* has sent out after transmitting the *m*th packet after time *t*. Assume that all *m* packets could be transmitted in the *k*th round. Equation (4), by replacing Acc_i^m with $DeficitCounter_i^0$ -$DeficitCounter_i^m$, is equivalent to

[046] $$\frac{DeficitCounter_i^m}{Quantum_i} = \frac{DeficitCounter_i^{m-1} - L_i^m}{Quantum_i}, \qquad (5)$$

[047] where $DeficitCounter_i^m$ denotes the residual quantum of flow *i* in this round after it puts the *m*th packet into the *Pre-order Queuing*. To further illustrate the equivalence, the following definition is provided:

[048] *Definition 1*: The Quantum Availability, QA_i^m, of the packet P_i^m is the ratio of its $DeficitCounter_i^m$ to $Quantum_i$, i.e.

[049] $$QA_i^m = \frac{DeficitConter_i^m}{Quantum_i}. \qquad (6)$$

[050] *Lemma 1:* For any packet P_i^m, its Quantum Availability QA_i^m satisfies

[051] $0 \le QA_i^m < 1$.

[052] *Lemma 2:* For the packet with the smallest timestamp in one round, its *QA* is the *largest*.

[053] Accordingly, the server sub-module 118 selects the packet with the largest *QA* within one round to send out. However, to avoid having to search for the packet with the largest *QA* among all packets that could be sent out in a round, classifier sub-module 114 classifies packets into several classes according to their *QA* and places them into the corresponding priority queues, i.e., priority queues 116_1-116_z.

[054] There are Z priority queues and, hence, Z classes. For the mth packet of flow i that can be sent out in this round, its class n_i^m can be derived as

[055] $$n_i^m = Z - \lfloor QA_i^m \times Z \rfloor = Z - \left\lfloor \frac{DeficitCounter_i^m}{Pqg_i} \right\rfloor, \qquad (7)$$

[056] where $DeficitCounter_i^m$ denotes the residual credits in byte for flow i in the kth round after the mth packet is placed into a priority queue, and Pqg_i denotes the granularity of priority queue for flow i derived as

[057] $$Pqg_i = \frac{Quantum_i}{Z}. \qquad (8)$$

[058] Below is one example of pseudo-code called "*PKT_Departure*" which may be used by packet departure module 110.

```
                PKT_Departure module
While(TRUE)
{ If MH_Empty() Then          // Imply no packets can be placed into PQ
  { Round_sys←Round_sys+ 1    // Announce arrival of the new round
    EC←WaitEvents(EV_m // wait until a packet was placed in PQ or
    inheap, EV_actlist)       //  any Fq's
    If ( EC = EV_actlist)     // Imply there are packets without sending out at last
    Then                      round
    { NumAckList←NumItem(AckList)  // There are NumAckList non-empty
                                   //  Fq's at last round
      While(NumAckList>0)     //For non-empty Fq's at last round,
      { I←Dequeue(AckList)    accumulate
        DC_I←DC_I + Quantum_I //  their residual credits for using at this
                              round
        SendMsg(PKT_Pass, I) // Ack PKT_Pass to pass packets of Fq_I into
                             PQ.
        NumAckList←NumAckList-1
      }
      WaitEvent(EV_minheap)   // Pause if no packets in PQ
    } // IF EC=EV_actlist
  }   // IF MH_EMPTY
  WaitEvent(ServerIdle)       // Pause if the server is sending out a packet
  MH_Lock()  //To avoid the MinHeapRoot being modified as MH_Delete() is
             involved.
  If Empty(Pq_MinHeapRoot) Then MH_Delete()
  MH_Unlock()
  Send(Dequeue(Pq_MinHeapRoot))  // Send out the packet in the Pq_I with the
                                   smallest I
                                 //  among non-empty Pq_I
```

[059] Tables 2 and 3 provide some definitions used by the pseudo-code modules described above.

TABLE 2

THE OPERATIONS USED IN *PKT_ARRIVAL* AND *PKT_DEPARTURE* AND *PKT_PASS* PROCEDURES

Operation	Description
Enqueue, Dequeue, NonEmpty, Empty, Head	The standard *Queue* operations
NumItem(A)	Return the number of entries in the *A*
MH_Insert(x),MH_Delete(), MH_Empty()	The standard operations of the min heap
MinHeapRoot	The minimum value among nodes of the min heap
MH_Lock, MH_Unlock	After locking, only one module can access the min heap
SetEvent(ev)	Signal the event *ev* The event will remain until signaled someone releases it
EventCase=WaitEvents(ev1,ev2, .)	Once any event is in the signaled state, return it to *EventCase* and release it Note ev1 is prior to ev2
SendMsg(x,y)	Send message along with value y to x
y=WaitMsg()	Wait message and store the received value in y

TABLE 3

THE VARIABLES USED IN *PKT_ARRIVAL* AND *PKT_DEPARTURE* AND *PKT_PASS* PROCEDURES

Variable	Description
Fq_i	The queue of flow *i*, *i*=1 *N*
Pq_j	Priority queue *j*, *j*=1 *Z*
$Quantum_i$	The created allocated to flow *i* in one round
DC_i	The *DeficitCounter* of that flow *i*
Pgg_i	The granularity of priority queue for flow *I*
Z	The number of priority queues
$Round_{sys}$	The identification of the system current round
$Round_i$	The identification of the round of flow I
$EV_{minheap}$	A event signaled as one key was placed into the empty min heap
$EV_{actlist}$	A event signaled as one item was placed into the empty ActList

[060] Fig. 2 shows a method for scheduling packets in accordance with the principles of the present invention. In step 200, input port 102 receives a packet and provides the packet to packet arrival module 106. In step 202, packet arrival module 106 identifies the packet's flow. In step 204, packet arrival module 106 determines whether the packet's flow is a new flow or a pre-existing flow. For example, packet arrival module 106 may work in conjunction with processor 104 to check service list 120. If the flow is not found within service list 120, e.g., by processor 104, then processing flows to step 206 in which packet arrival module 106 sends a message to packet departure module 110 to add the new flow to service list 120. Packet arrival module 104 may then add a flow queue to flow queues 112_1-112_n for the new flow. Processing then proceeds to step 208 as described below.

[061] If the flow is found within service list 120, then processing flow to step 208. In step 208, packet arrival module 106 places the packet in its corresponding

flow queue, i.e., one of flow queues 112_1-112_n and sends a message to pre-order queuing module 108.

[062] In step 210, upon receiving the message, pre-order queuing module 108 notifies classifier sub-module 114 to retrieve and classify the packet from flow queues 112_1-112_n and classifies the packet. Classifier sub-module 114 may classify the packet (and its flow) in a wide variety of ways. For example, classifier sub-module 114 may classify the packet based upon: a source address, a destination address, or other information such as a service class (e.g., constant bit rate), transport control protocol port, etc. Other information may also be used by classifier sub-module 114 to classify the packet.

[063] In step 212, classifier sub-module 114 places the packet in priority queues 116_1-116_z within priority queuing sub-module 115 based upon the packet's classification. Upon the placement of the packet in priority queues 116_1-116_z, pre-order queuing module 108 sends a message to packet departure module 110. The message may include the size of the packet and the priority queue, i.e., one of priority queues 116_1-116_z, into which the packet was placed. Processing then returns to step 200, where the process for scheduling packets repeats.

[064] Fig. 3 shows a method of transmitting packets in accordance with the principles of the present invention. In particular, in step 300, server sub-module 118 sends a message announcing a new round, e.g., to pre-order queuing module 108 via processor 104. Server sub-module 118 may announce a new round at various times, e.g., upon packet departure module 110 receiving a message that there are packets in priority queuing sub-module 115 and/or at pre-determined intervals.

[065] In step 302, server sub-module 118 determines which priority queues among priority queues 116_1-116_z are non-empty (i.e., have packets placed within them). Server sub-module 118 may determine the non-empty queues by searching service list 120.

[066] In step 304, server sub-module 118 confirms that there are non-empty priority queues. If all priority queues 116_1-116_z are empty, then processing proceeds to step 306. Otherwise, processing proceeds to step 308. In step 306, server sub-module 118 waits to announce a new round. Server sub-module 118 may wait to receive a message from pre-order queuing module 108. Alternatively, server sub-module 118 may wait for a predetermined period of time. However, any type of wait period and/or initiation of a new round is within the principles of the present invention.

[067] In step 308, server sub-module 118 determines whether the current round is complete. For example, a round may be complete upon server sub-module 118 servicing all non-empty priority queues. If the round is complete, then processing flows back to step 300 where a new round is announced.

[068] If the round is not complete, then processing flows to step 310 in which the server sub-module 118 determines a priority queue to service next. Server sub-module 118 may determine the next priority queue to service such that the priority queue among priority queues 116_1-116_z with the highest priority is serviced before serving lower priority queues. Alternatively, server sub-module 118 may service priority queues 116_1-116_z in a rotating fashion.

[069] In one embodiment, a complete binary tree (called min_heap in the pseudo code), is used to enable server sub-module 118 to efficiently determine which non-empty priority queue among priority queues 116_1-116_z has the highest priority. For example, referring now to the pseudo code PKT_Pass and PKT_Departure described above, once packets are placed into priority queuing sub-module 115 (e.g., by PKT_Pass) the status of the event $EV_{minheap}$ is signaled to notify packet departure module 110 (e.g., PKT_Departure) to send out a packet. After transmission is complete, ServerIdle is signaled and PKT_Departure repeats the last action until the $Pq_{MinHeapRoot}$ is empty. The function MH_Delete may then delete the root node of the binary tree (i.e., min_heap) and set MinHeapRoot to the smallest j among residual nodes. When the min_ heap is empty, i.e., all Pq_j's are empty, PKT_Departure may declare the arrival of a new round by adding 1 to $Round_{sys}$. For all non-empty Fq's, i.e., flows with packets remaining in the last round, PKT_Departure may update DeficitCounters according to the sequence in the service list (e.g., AckList) and request that PKT_Pass classify the eligible packets into the priority queuing sub-module 115.

[070] In step 312, server sub-module 118 calculates the accumulated quantum (i.e., as indicated by DeficitCounter) for the packet. In step 314, server sub-module 118 then determines whether the packet will be sent out during the current round. For example, If the packet's size is smaller than $DeficitCounter_f^j$, server sub-module 118 decreases $DeficitCounter_f^j$ by the packet size and processing proceeds to step 316 where server sub-module 118 sends the packet out, e.g., via output port 122. Alternatively, server sub-module 118 may reset $DeficitCounter_f^j$ to

zero, i.e., the residual quantum remaining from the previous round cannot be carried over to avoid delaying service to other priority queues. In addition, during progress of a particular round, if a packet arrives in a priority queue having a higher priority than the priority queue currently being serviced, then server sub-module 118 may service the higher priority queue out of order within the round and send the packet out before any other packets from lower priority queues are sent out.

[071] If the packet's size is larger than $DeficitCounter_i^j$, then processing proceeds to step 318 where server sub-module 118 may hold the packet until a subsequent round. Server sub-module 118 may repeatedly hold the packet until the size of the head packet is larger than $DeficitCounter_i^j$, i.e., there is insufficient residual quantum to serve a subsequent packet, or there are no remaining packets in the priority queue. During a subsequent round, the next time the priority queue gets its turn, server sub-module 118 may send out additional $DeficitCounter_i^j$ bytes of data in addition to $quantum_i$ bytes.

[072] Fig. 4a illustrates the operation of Deficit Round Robin (DRR) in comparison with weighted fair queuing (WFQ). As shown in Fig. 4a, there are four flows requesting the same amount of bandwidth and having fixed, but heterogeneous packet sizes. The same quantum is assigned to all of them and, according to the known DRR algorithms, the quantum should be equal to the largest maximum packet size among all flows. Packets 1, 4, 6, and B arrive at the same time and all have greedy flow sources, i.e., all flows are heavily backlogged. By comparing the output pattern in DRR with that in WFQ shown in Fig. 4a, three problems may be observed. First, packets 1, 4 and 6 were transmitted under DRR

out of sequence (i.e., in comparison to WFQ such as 6, 1 and 4) since DRR only considers whether a packet could be sent out in a round and does not consider eligible transmission sequence for packets. Second, packets 6, 7, 8 and 9 are sent out in a batch under DRR, which in terms of latency and fairness is not considered good behavior in a packet switched network. Third, the transmission time of packet B (with a size slightly greater than the residual credits of the first round) is delayed under DRR until the next round, i.e., after all other flows finish their transmissions in the second round, which may be too long a delay. Under DRR, the delay increases with the frame size and a larger quantum size produces larger frame size.

[073] Fig. 4b shows an embodiment consistent with the present invention using Pre-Order Deficit Round Robin and operating on the same input pattern and assumptions used in Fig. 4a. In this example, packets are classified into 4 classes, so that $Z = 4$. Assuming for all flows the quantums are equal to 400 and the size of packet B is 500, then packet B cannot be sent out in the first round. However, in the next round, $DeficitCounter^{B}$ would be equal to 300, i.e. 400+400-500. In accordance with the present invention, the packet would then be classified into the first class, i.e., $4-\lfloor 300/(400/4)\rfloor$ and could be sent out at the beginning of the next round. Other packets are put into the priority queuing sub-module 115 according to the same rule. Thus, as shown in Fig. 4b, even under the assumption that all flows are heavily backlogged and there are enough priority queues, embodiments consistent with the present invention exhibit good performance in comparison to the typical DRR performance.

[074] Below, the performance of PDRR is analyzed in terms of delay bound and throughput fairness. Under the analysis below, PDRR is shown to have an O(1) per-packet time complexity in most case. In particular, consider a queuing system with a single server of rate C.

[075] *Definition 2*: *A backlogged period for flow i* is any period of time during which flow *i* is continuously backlogged in the system. Time t_0 is the beginning of a backlogged period of flow *i* and t_k indicates the time that the *k*th round in PDRR is completed. $W_i(\tau, t_k)$ denotes the service offered to flow *i* in the interval $(\tau, t_k]$ by the server and L_i is the maximum packet size of flow *i*.

[076] *Lemma 3*: Under PDRR, if flow *i* is continuously backlogged in the interval $(t_0, t_k]$ then at the end of the *k*th round,

[077] $$W_i(t_0, t_k) \geq k\phi_i - D_i^k, \quad (9)$$

[078] where D_i^k is the value of the *DeficitCounter*$_i$ at the end of the *k*th round and ϕ_i is *Quantum*$_i$.

[079] *Lemma 4*: Let t_0 be the beginning of a backlogged period of flow *i* in PDRR. At any time *t* during the backlogged period,

[080] $$W_i(t_0, t) \geq \max(0, r_i(t - t_0 - \frac{(2 + \frac{1}{Z})F - \phi_i}{C})), \quad (10)$$

[081] where *F* is equal to $\sum_{1}^{\#\,flow} \phi_i$, *Z* is the number of priority queues, and r_i is the rate allocated to the flow *i*.

[082] *Theorem 1*: The PDRR server belongs to LR with latency θ^{PDRR} less than or equal to

[083] $\frac{(2+\frac{1}{Z})F-\phi_i}{C}$. (11)

[084] According to equation (1), replacing *F* with $\phi_i C/r_i$ in equation (11) results in

[085] $\theta^{PDRR} \le (2+\frac{1}{Z})\frac{\phi_i}{r_i}-\frac{2\phi_i}{C}$. (12)

[086] Equation (11) shows that PDRR improves latency, as opposed to the DRR whose latency is $(3F-2\phi_i)/C$. Furthermore, in the worst case, if the form of θ^{SCFQ} is translated to the form of θ^{PDRR}, the latency of PDRR is shown to be similar to that of SCFQ, which is $(2F-\phi_i)/C$. Equation (12) demonstrates that the latency of PDRR is inversely dependent with the allocated bandwidth, and independent of the number of active flows.

[087] *Theorem 2*: The scheduling algorithm at the server is PDRR and the traffic of flow *i* conforms to a leaky bucket with parameters (σ_i,ρ_i), where σ_i and ρ_i denote the burstiness and average rate of the flow *i*, respectively. The rate allocated to the flow *i* is assumed to be equal to ρ_i. If $Delay_i$ is the delay of any packet of flow i, then

[088] $Delay_i \le \frac{\sigma_i}{\rho_i}+\frac{(2+\frac{1}{Z})F-\phi_i}{C}$. (13)

[089] *Theorem 3*: For a PDRR scheduler,

[090] $Fairness^{PDRR} = \frac{(2+\frac{1}{Z})F}{C}$, (14)

[091] where $Fairness^{PDRR}$ is the fairness of the server PDRR. Thus, $Fairness^{PDRR}$ is smaller than $Fairness^{DRR}$, which is $3F/C$.

[092] As noted in the analysis above, for each packet, PKT_Arrival inserts the packet into its corresponding Fq and PKT_Pass takes the packet from its Fq to the Pq_j where j is found in a constant number of operations. PKT_Departure repeatedly picks a packet from the $Pq_{MinHeapRoot}$ whose MinHeapRoot always

TABLE 4.
THE TRAFFIC PARAMETERS AND *QUANTUM* SIZE OF TWO GROUPS

Group	Traffic Type	Bit Rate (Mbps)	Packet Size (byte)	Quantum Size (byte)
GA	CBR	4	50	500
GB	CBR	4	500	500

presents the smallest j among non-empty Pq_j's. As the min heap operations are not invoked under the assumption that each accessed Pq is non-empty, all the complexities of the above operations are O(1). When the $Pq_{MinHeapRoot}$ is empty, a delete operation of the min heap is invoked by PKT_Departure to get the new MinHeapRoot. The reheapification loop has time complexity O(log Z), where Z is the maximum number of keys present in the "min heap", i.e., the number of non-empty *Pq*'s at that moment. A similar situation also occurs as *PKT_Pass* must insert a new *j* into the min heap. Above, Table 4 summarizes the complexity of PKT_Arrival, PKT_Pass, PKT_Departure when a non-empty or empty priority queue is accessed.

[093] However, embodiments consistent with the present invention allow for packets to be sent out individually and have low delay possibility by operations of the min heap. First, the min heap operation may be involved when the accessed *Pq* is empty. In contrast, sorted-priority algorithms, e.g., SCFQ and WFQ, use two operations, insert and delete, repeatedly when a packet is being sent out. Secondly, embodiments consistent with the present invention allow for the scalar of the min heap to be small such that the maximum number of keys is the number of *Pq*'s instead of the number of flows. Moreover, embodiments consistent with the present

invention allow concurrent insertions and deletions on the heap. According to the principles of the present invention, *PKT_Departure*, after getting the next smallest value *j* immediately via one comparison between the two leaf keys of the root, can start to send out packets in the Pq_j concurrently during the period of the reheapification loop. Therefore, the time complexity of PDRR in accordance with the present invention is O(1) in most cases and O(log Z) in some special cases. Accordingly, embodiments consistent with the present invention utilize an algorithm with a lower complexity than algorithms such as SCFQ, which requires O(log N) operations where N is the number of flows.

[094] Figs. 5-8 show various simulation results to compare the performance of embodiments consistent with the present invention using PDRR, with DRR and SCFQ. For the simulation, the link bandwidth, i.e., server capacity, was assumed to be 80 Mbps, shared by 20 flows. The 20 flows were divided into two groups, GA and GB. The following experiment was performed to show the problem of bursty transmission by DRR and the improvement effected by PDRR. The traffic sources were assumed to be Constant Bit Rate (CBR) with fixed packet size. Below, Table 5 indicates the traffic parameters of the two groups, GA and GB.

TABLE 5.

Operation / Case	***PKT_ARRIVAL***	***PKT_PASS***	***PKT_DEPARTURE***
Pq non-empty	O(1)	O(1)	O(1)
Pq empty	O(1)	O(log Z)	O(log Z)

[095] Furthermore, the bit rates of GA and GB were set equal, the maximum packet size among both groups was 500 bytes, and the same quantum size of 500 bytes was allocated to them. The packet arrival rate of GA flows was 10 times the

size of GB. In this experiment, 10 priority queues were used. The delay time of a particular GA flow under DRR, PDRR, and SCFQ, respectively were measured.

[096] As shown in Fig. 5, packets of flow within DRR are sent out in a batch once the flow is served. In PDRR the flow is able to spend its quantum in several pieces, so that packets could be sent out uniformly. Another observation is that packets in SCFQ suffer high delay jitter, which is due to GA flows having a packet arrival rate ten times that of the GB flow. Further, for GA packets that arrive while the server is serving a large packet of GB, their virtual arrival times are equal, which causes the server not to send them out in their actual arrival sequence.

[097] As shown in Fig. 6, the GA flow in DRR has a larger average delay than that in PDRR. The traffic source was assumed to be shaped by a leaky bucket with on/off rates are both 1000 1/μsec and a bit rate of 4 Mbps. The GA flows were assigned a larger packet size than GB flows, however all flows requested the same bandwidth. With the increase of GA's packet size to that of GB's, the curve of PDRR in Fig. 6 shows that PDRR performs well, especially with heterogeneous traffic sources.

[098] Fig. 7 shows that as the ratio of GA's packet size to GB's packet size is increased, small packets in PDRR perform better than large packets, unlike the case of DRR. GA flows were assumed to have a higher bit rate than that of GB flows. Unlike DRR, PDRR considers the information provided in the quantum consumed by a packet and reorders the transmission sequence of packets in one round. Under DRR, a node only considers whether a packet could be sent out and ignores the transmission order of packets in one round. In a high flow environment with

heterogeneous bandwidth requirements, embodiments consistent with the present invention using PDRR perform better than DRR because pre-order queuing module 108 can transmit packets more uniformly within a round. In addition, Fig. 8 shows that the average delay of GA flows in PDRR is lower than that in DRR.

[099] Fig. 9 shows performance of an embodiment consistent with the present invention as the number of priority queues is varied for a specific traffic environment. As described above, embodiments consistent with the present invention enable a flow to use its quantum uniformly within a round, especially when its quantum is several times its maximum packet size. For example, for a flow i, pre-order queuing with $(Quantum_i / L_i)$ priority queues can reach the above goal. Thus, for a specific traffic environment, *Z* priority queues are sufficient where *Z* is equal to the maximum value of $(Quantum_i / L_i)$ among all flows, i.e. $\max_i(Quantum_i / L_i)$. Fig. 9 shows experimental data illustrating the relationship between *Z* and $\max_i (Quantum_i / L_i)$. All traffic sources were assumed as CBR with fixed packet size. Fig. 9 shows the average delay of the flow whose $(Quantum_i / L_i)$ equals $\max_i(Quantum_i / L_i)$. For each line, this flow receives the smallest average delay when $Z = \max_i(Quantum_i / L_i)$, which confirms the advantages realized through practice of the present invention.

[0100] The present invention may also apply to flows with different allocated bandwidth but having the same distribution of packet sizes. Traffic types, other than CBR and MMPP, are also within the principles of the present invention. Other embodiments of the invention will be apparent to those skilled in the art from

consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.

Appendix: Proofs of Primary Results

[0101] *Proof of Lemma 1*: This is proved by showing that for any packet P_i^m, its $DeficitCounter_i^m$ must be positive and smaller than $Quantum_i$. The value in *DeficitCounter* cannot be negative and could only increase by $Quantum_i$ in the *UpdateOneFlow* procedure. It is assumed that there is insufficient credit to send out the packet P_i^m, i.e., the $DeficitCounter_i^{m-1}$ is smaller than L_i^m, the size of this packet. After updating and sending out the packet P_i^m,

$DeficitCounter_i^m = DeficitCounter_i^{m-1} + Quantum_i - L_i^m$. As $DeficitCounter_i^{m-1} < L_i^m$, $DeficitCounter_i^m$ must be smaller than $Quantum_i$. As a result, lemma 1 is proved.

[0102] *Proof of lemma 2:* According to equation (6), equation (5) is equivalent to

[0103] $$QA_i^m = QA_i^{m-1} - \frac{L_i^m}{Quantum_i} \qquad (A.1)$$

[0104] Since $L_i^m > 0$, $Quantum_i > 0$ and $r_i > 0$, from equation (3) and equation (A.1) for any *m*

[0105] $TS_i^m > TS_i^{m-1}$ and $QA_i^{m-1} > QA_i^m$.

[0106] In the same round, for the packet P_i^m with the smallest timestamp, its *m* is smallest among all packets, and the QA_i^m of the packet with the smallest *m*. Thus, for the packet with the smallest timestamp in one round, its *QA* is the largest.

[0107] *Proof of Lemma 3*: PDRR only modifies the service sequence of the packets in a DRR round. Therefore, the packets in DRR that could be sent out in a round during a backlogged period, can still be sent out in the same PDRR round.

[0108] *Proof of Lemma 4*: For each time interval $(t_{k-1}, t_k]$,

[0109] $t_k - t_{k-1} \le \frac{1}{C}\left(F + \sum_{j=1}^{N} D_j^{k-1} - \sum_{j=1}^{N} D_j^k\right)$. (A.2)

[0110] By summing over *k-1*,

[0111] $t_{k-1} - t_0 \le (k-1)\frac{F}{C} + \frac{1}{C}\sum_{j=1}^{N} D_j^0 - \frac{1}{C}\sum_{j=1}^{N} D_j^{k-1}$. (A.3)

[0112] It is assumed there are two packets, P_i^A and P_i^B, in the Fq_i whose sizes are L_i^A and L_i^B, ($L_i^B = \phi_i$), respectively, and only P_i^A can be sent out at the (*k-1*)th round. All other flows exhaust their *DeficitCounter*. Thus, $D_i^{K-1} = \phi_i - \Delta$ where $0 < \Delta \le \phi_i$, $D_j^{k-1} = 0$ for $j \ne i$, and

[0113] $t_{k-1} - t_0 \le (k-1)\frac{F}{C} + \frac{F - \phi_i + \Delta}{C}$. (A.4)

[0114] Under this assumption, in the *k*th round P_i^B would be placed into the Pq_n where

[0115]
$$n = Z - \left\lfloor \frac{D_i^K}{Pqg_i} \right\rfloor = Z - \left\lfloor \frac{D_i^{K-1} + \phi_i - L_i^B}{\phi_i / Z} \right\rfloor = Z - \left\lfloor \frac{D_i^{K-1}}{\phi_i} Z \right\rfloor = Z - \left\lfloor \frac{\phi_i - \Delta}{\phi_i} Z \right\rfloor = Z - \left\lfloor Z - \frac{\Delta}{\phi_i} Z \right\rfloor = \left\lceil \frac{\Delta}{\phi_i} Z \right\rceil \quad \text{(A.5)}$$

[0116] and the maximum amount of data that could be served before P_i^B is $((n/Z)F - \phi_i)$. Thus, for any time *t* from the beginning of the *k*th round until the P_i^B is served,

[0117]
$$t - t_0 \le \frac{\frac{n}{Z}F - \phi_i}{C} + t_{k-1} - t_0 \le (k-1)\frac{F}{C} + \frac{F - \phi_i + \Delta}{C} + \frac{\frac{n}{Z}F - \phi_i}{C}, \quad \text{(A.6)}$$

[0118] or equivalently,

[0119] $k-1 \geq \frac{(t-t_0)C}{F}+\frac{2\phi_i-\Delta}{F}-\frac{n}{Z}-1$. (A.7)

[0120] Replacing *k* with *k*-1 in equation (9) and defining r_i, the reserved rate of flow *i*, as $(\phi_i C/F)$, results in

[0121]
$$\begin{aligned} W_i(t_0,t_{k-1}) &\geq \phi_i\left(\frac{(t-t_0)C}{F}+\frac{2\phi_i-\Delta}{F}-\frac{n}{Z}-1\right)-D_i^{k-1} \\ &= r_i\left(t-t_0+\frac{2\phi_i-\Delta}{C}-\frac{n}{Z}\times\frac{F}{C}-\frac{F}{C}-\frac{D_i^{k-1}}{r_i}\right) \\ &= r_i\left(t-t_0-\left(1+\frac{n}{Z}+\frac{D_i^{k-1}}{\phi_i}\right)\frac{F}{C}+\frac{2\phi_i-\Delta}{C}\right) \\ &= r_i\left(t-t_0-\frac{\left(1+\frac{n}{Z}+\frac{D_i^{k-1}}{\phi_i}\right)F-2\phi_i+\Delta}{C}\right) \end{aligned} \quad \text{(A.8)}$$

[0122] Replacing D_i^{k-1} with $\phi_i-\Delta$ and since $\frac{n}{Z}-\frac{\Delta}{\phi_i}=\left\lceil\frac{\Delta}{\phi_i}Z\right\rceil\frac{1}{Z}-\frac{\Delta}{\phi_i}\leq\frac{1}{Z}$ for any Δ,

[0123]
$$\begin{aligned} W_i(t_0,t_{k-1}) &\geq r_i\left(t-t_0-\frac{\left(2+\frac{n}{Z}-\frac{\Delta}{\phi_i}\right)F-2\phi_i+\Delta}{C}\right) \\ &\geq r_i\left(t-t_0-\frac{\left(2+\frac{1}{Z}\right)F-\phi_i}{C}\right). \end{aligned} \quad \text{(A.9)}$$

[0124] In the worst case, flow *i* was the last to be updated during the *k*th round and its packet L_i is inserted into the tail of Pq_n. The following two cases are considered:

[0125] **Case 1:** At time *t before* the time that flow *i* is served in the *k*th round, i.e.

[0126] $t_{k-1}<t\leq t_{k-1}+\frac{n}{Z}F-\phi_i$,

[0127] there results

[0128] $W_i(t_0,t)=W_i(t_0,t_{k-1})$. (A.10)

[0129] **Case 2:** At time t *after* the time that flow *i* starts being served in the *k*th round, i.e.

[0130] $t_{k-1}+\frac{n}{Z}F-\phi_i<t\le t_{k_i}$,

[0131] there results

[0132] $W_i(t_0,t)=W_i(t_0,t_{k-1})+W_i(t_{k-1},t)\ge W_i(t_0,t_{k-1})$. (A.11)

[0133] Thus, for any time *t*,

[0134] $W_i(t_0,t)\ge \max(0,r_i(t-t_0-\frac{(2+\frac{1}{Z})F-\phi_i}{C}))$. (A.12)

[0135] *Proof of Theorem 3*: The beginning of the *k*th round is assumed and there are two packets P_i^A and P_i^B in the Fq_i whose sizes are L_i^A and ϕ_i, respectively, and only P_i^A can be sent out at the *k*th round. The packet P_i^B's class is *n*, which implies it will enter the *n*th priority queue. In the (*k+1*) round, all packets of another flow *j* whose classes are larger than *n,* could not be sent out before the time *t* when P_i^B is sent out, as the server always selects packets from the nonempty Pq_j with the smallest *j*. Thus, before *t*, for another flow *j*,

[0136] $W_j(t_0,t)\le k\phi_j+\frac{n}{Z}\phi_j$. (A.13)

[0137] Also as $t_k<t<t_{k+1}$, from equation (9) for flow *i*,

[0138] $W_i(t_0,t)\ge (k-1)\phi_i+\Delta$. (A.14)

[0139] From equations (A.5), (A.13) and (A.14) it is concluded that

[0140] $$\left|\frac{W_i(t_0,t)}{r_i}-\frac{W_j(t_0,t)}{r_j}\right|\le (k+\frac{n}{Z})\frac{\phi_j}{r_j}-(k-1)\frac{\phi_i}{r_i}-\frac{\Delta}{r_i} \le (1+\frac{n}{Z}-\frac{\Delta}{\phi_i})\frac{F}{C}\le (1+\frac{1}{Z})\frac{F}{C}. \quad \text{(A.15)}$$

[0141] This bound applies to time intervals that began at t_0. For any arbitrary interval,

[0142] $$\left|\frac{W_i(t_1,t_2)}{r_i}-\frac{W_j(t_1,t_2)}{r_j}\right|\le(2+\frac{1}{Z})\frac{F}{C}. \qquad \text{(A.16)}$$

[0143] Thus, for any two flows i and j,

[0144] $$Fairness^{PDRR}=\frac{\left(2+\frac{1}{Z}\right)F}{C}. \qquad \text{(A.17)}$$